RMIT UNIVERSITY

Software Engineering
Web Programming

Full-Stack Web Application Online Learning Platform

Student Name: Nguyen Hong Anh

Student ID: s3924711

Date: 21/09/2024

Contents

Introduction

The Online Learning Platform is designed to facilitate an engaging and accessible environment for both learners and instructors. This project aims to bridge the gap between educational content and users by providing an intuitive interface that supports course management and user interactions. With a focus on user experience and functionality, the platform serves as a comprehensive solution for online education.

About the group project

The project was initiated to address the increasing demand for online learning solutions, especially in a post-pandemic world where traditional education methods have been challenged. The platform allows:

- **Instructors** to create and manage courses, connect with learners, and showcase their expertise.
- **Learners** to easily browse, enroll in, and participate in courses that fit their interests and needs.
- **Administrators** to oversee the platform, manage users, and ensure a smooth operation.

Objectives:

- Implementing a robust user authentication system for both instructors and learners to ensure secure access.
- Facilitating easy navigation and interaction within the platform, including course browsing, enrollment, and profile management.
- Utilizing modern web technologies (Node.js, Express, and MongoDB) to enhance the platform's performance and scalability.

About the team

This project was developed solo by Nguyen Hong Anh, who took on multiple roles throughout the development process, including:

- **Frontend Developer:** Designed and implemented the user interface, ensuring responsive design and enhancing user experience using HTML, CSS, and JavaScript.
- **Backend Developer:** Developed server-side logic, managed database interactions, and implemented user authentication using Node.js and Express.
- **Database Administrator:** Set up and maintained the database using MongoDB ensuring data integrity and efficient data handling.

Development Process:

The project was executed using various tools and methodologies to ensure effective development:

- **Version Control:** Git was used for source code management, allowing for organized code tracking.
- **Project Management:** Tools like Microsoft Planner were utilized to set tasks, manage deadlines, and track progress.

Requirement analysis

Functional Requirements

Functional requirements specify what the system must be capable of performing [1]. For the Online Learning Platform, the following functional requirements were identified:

1. User Authentication
    - **Registration:** Users must be able to create an account by providing essential information.
    - **Login/Logout:** Users should be able to log in using their credentials and securely log out of the system.
    - **Password Recovery:** Users must have the ability to reset their password through a "Forgot Password" feature.
2. User Roles
    - **Role Definition:** The system must distinguish between different user roles: Admin, Instructor, and Learner.
    - **Role-Based Access Control:** Different functionalities should be available based on the user role. For example:
        - Instructors can create, edit, and delete their courses.
        - Learners can browse, enroll in courses, and view details about that course.
        - Admins can manage all users and courses and access overall system settings.
3. Course Management
    - **Course Creation:** Instructors should be able to create courses by entering details such as course name, description, price, and their categories.
    - **Course Editing and Deletion:** Instructors must have the ability to modify or remove courses as needed.
    - **Course Browsing:** Learners should be able to view a list of available courses, search for courses by name or category, and filter results.

4. Enrollment Management
    - **Enroll in Courses:** Learners should be able to enroll in available courses, with confirmation notifications.
    - **View Enrolled Courses:** Learners must be able to view a list of courses they are enrolled in and access course materials.
5. FAQs and Support
    - **FAQs Section**: A dedicated page for frequently asked questions to help users find answers quickly.
    - **Contact Support:** A form for users to submit inquiries or issues they encounter.

Non-Functional Requirements

Non-functional requirements describe how the system performs certain functions [1]. The following non-functional requirements were identified for the Online Learning Platform:

1. Security
    - User data must be protected through secure authentication processes, including password encryption using bcrypt.
    - Implement HTTPS to ensure secure communication between clients and the server.
2. Usability
    - The platform should have a user-friendly user interface that allows the user to navigate easily.
3. Maintainability
    - The code should be well-organized and documented to facilitate future updates and maintenance
    - Utilize git to track changes and manage development processes.

System design

Architecture Overview

The Online Learning Platform follows a client-server architecture where the clients (front-end) interact with the server (back-end) to perform various operations. The back-end is responsible for data management and authentication, while the front-end provides a user friendly interface.

- **Client:** The front-end application developed using HTML, CSS, EJS and JavaScript.

- **Server:** The backend is built with Node.js and Express.
- **Database:** MongoDB is used as the primary database for user data.

Database Design

- MongoDB Collections:
    - User collection: Store user information. (e.g., email, password, role)
    - Course collection: Store course information. (e.g., course name, description)

Example MongoDB schema:

```
const userSchema = new Schema({
  fullName: { type: String, required: true },
  email: { type: String, required: true, unique: true, lowercase: true,
trim: true },
  phoneNumber: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  address: { type: String, default: '' },
  city: { type: String, default: '' },
  zipcode: { type: String, default: '' },
  country: { type: String, default: '' },
  role: { type: String, enum: ['admin', 'instructor', 'user'], default:
'user' },
  schoolName: { type: String, default: '' },
  jobTitle: { type: String, default: '' },
  specialization: { type: [String], default: [] },
  courses: [{ type: String }] // Change to an array of strings
}


const courseSchema = new Schema({
  courseName: { type: String, required: true },
  price: { type: Number, required: true },
  description: { type: String, default: '' },
  categories: { type: [String], default: [] }, // Add categories field
  createdAt: { type: Date, default: Date.now }
});
```

Security Measures

- **Data Encryption:** Passwords must be hashed using bcrypt before storing them in the database.
- **HTTPS:** The platform should be served over HTTPS to encrypt data in transit.

Source code management

Effective source code management is crucial for maintaining the integrity and quality of a software project. For your Online Learning Platform, the following practices and tools were employed to manage the source code:

1. Version Control System
    - **Git:** The project utilizes Git as the version control system. This allows for tracking changes, managing multiple versions of the code.
    - **Repository Hosting:** The code is hosted on GitHub, enabling easy access, and issue tracking.
2. Commit Practices
    - **Descriptive Commit Messages:** Each commit includes a clear and descriptive message that explains the changes made.
3. Pull Requests
    - **Code Review Process:** Before merging changes into the main branch, pull requests are created. This allows to review the code, suggest improvements, and ensure code quality.
4. Documentation
    - **README.md:** The repository includes a README.md file that provides an overview of the project, setup instructions, and usage examples.

HTML (HyperText Markup Language)

HTML serves as the backbone of the web application, structuring the content and providing structure through semantic elements.

Structural Elements:

- **<header>**: This element contains the platform's logo and the main navigation bar. It helps define the top section of the page and provides context for the content that follows.
- **<footer>**: Positioned at the bottom, the footer includes secondary navigation links and copyright information
- **<main>**: The main content area where the primary information is displayed. This tag signifies the central part of the page and contains the most important content.
- **<section>**: Used to group related content within the main area.
- **<div>**: Utilized for generic grouping of elements, providing flexibility for layout and styling without implying any specific meaning.

Form Elements:

- **<form>**: Used for user input, such as registration and contact forms. Each form captures specific data from users and is essential for user interactions.
- **<input>**: Various input types (text, email, and password) facilitate user data entry, ensuring a smooth registration and login experience.
- **<select>**: Dropdown menus allow users to select from predefined options, such as choosing an account type or country.
- **<textarea>**: Multi-line text inputs are provided for longer user messages, such as feedback in the contact form.

Navigation Elements:

- **<nav>**: This element organizes the navigation links, making it clear where users can go within the application. The links within the nav help users access different sections like Home, About Us, and FAQs.

CSS (Casading Style Sheets)

Bootstrap Integration:

- Bootstrap is employed for responsive design and layout. Its grid system and pre-defined classes simplify the creation of consistent styles across different devices.

Custom Styles:

- Custom CSS provides a unique look and feel, including styles for headers, footers, and sections.
- **@font-face**: Allows the inclusion of custom fonts (e.g., Themify Icons) to enhance iconography.
- Global styles reset margins and paddings for uniformity:

```css
* {
  padding: 0;
  margin: 0;
  box-sizing: border-box;
}
```

Navigation Style

The navigation bar is styled for clarity and usability, with hover effects to improve interactivity:

```css
.custom-navbar {
  background-color: black;
}

.custom-navbar .nav-link:hover {
  color: #ccc; /* Lighter color on hover */
}
```

Modal Styling

Custom styles for modals ensure they are visually appealing and user-friendly. For example, the modal backdrop has a semi-transparent background to enhance focus on the modal content:

```css
.modal {
  background-color: rgba(0, 0, 0, 0.7); /* Black background with opacity
*/
}
```

JavaScript

Account Type Selection

The account type selection feature allows users to choose between different account types (e.g., user or instructor). When an instructor account is selected, additional fields relevant to instructors are displayed. This dynamic behavior is achieved using event listeners that toggle the visibility of the instructor-specific fields based on user input.

Key Code Sections:

- Event listeners on radio buttons to control visibility.
- Initialization to ensure correct display based on the selected option.

Modal Management for Terms of Service and Privacy Policy

This section manages the display of modals containing the terms of service and privacy policy. Users can open these modals by clicking on respective links and close them using close buttons named “Agree” or by clicking outside the modal area.

Key Code Sections:

- Functions to open and close modals.
- Event listeners to handle modal interactions.
- Dynamic handling of multiple modals and their close buttons.

Member Detail Modals

Detailed member information is displayed in modals triggered by clicking on images. Each image is linked to a specific modal, which opens upon interaction, allowing users to view additional information about a member.

Key Code Sections:

- Event listeners on images for modal activation.
- Close functionality for modals.

Contact Form Validation

The contact form validation ensures that user input is correct before submission. The validation checks include name length, email format, optional phone number validation, selection of contact days, and message length requirements. If validation passes, the form is able to be submitted; otherwise, relevant alerts are displayed to guide user correction.

Key Code Sections:

- Validation logic for different input fields (name, email, phone, message).
- Use of regular expressions for email validation.
- User feedback through alerts for form submission success or validation errors.

Data Management

Introduction

Data management in this application encompasses how user data is collected, validated, and potentially stored upon form submission. The JavaScript code is responsible for ensuring data integrity through validation mechanisms and handling user interactions efficiently.

Data Validation

Data validation is a critical component of data management, ensuring that the information collected from users meets predefined criteria before being processed or stored. The validation checks include:

Key Validation Criteria:

- **Name:** Must be at least 3 characters long to prevent invalid entries.
- **Email:** Must match a regular expression pattern to ensure correct email formatting.
- **Phone Number:** Validated only if the user selects "phone" as their preferred contact method.
- **Contact Days:** At least one day must be selected from the available options.
- **Message:** Length must be between 50 and 500 characters to ensure meaningful input.

User Input Collection

User input is collected through various forms, including registration and contact forms. These forms consist of multiple fields, each designed to capture specific information required for user accounts or inquiries.

Key Aspects:

- **Input Fields:** Users provide data through text inputs (e.g., name, email, phone), radio buttons (e.g., account type), checkboxes (e.g., preferred contact days), and text areas (e.g., message, feedback).

- **Dynamic Fields:** For instructors, additional fields are dynamically displayed based on user selection, ensuring relevant information is captured.

Form Submission

Once validation is successful, the application processes the form submission. While the current implementation uses alert to confirm successful submission, this section can be expanded to include actual data handling such as sending the data to a server or database for storage.

Key Considerations:

- **Data Format:** Consideration for how data will be structured when sent to the server (e.g., JSON format).

- **Asynchronous Handling:** Potential use of fetch to send form data asynchronously without reloading the page.

- **Error Handling:** Implementing try-catch blocks to manage errors during data submission, ensuring a smooth user experience.

Result

Data Accuracy and Integrity

- **Validation Success Rate:** Implementation of strict input validation significantly reduced the amount of incorrect or incomplete data submitted by users.
- **Contact Information Accuracy:** Enhanced validation for email and phone fields contributed to improved data quality.

Code Quality and Maintainability

- **Source Code Management:** The use of version control allowing for seamless integration of code contributions and regular code reviews.
- **Modular Code Structure:** The implementation of modular JavaScript code enhanced readability and maintainability, supporting future updates and feature additions.

Conclusion

The project successfully achieved its objectives, demonstrating the viability of the web application in providing a user-friendly experience for managing user registrations and inquiries. Overall, the project showcases a commitment to quality and user satisfaction. Ongoing improvements and adaptations will be crucial to maintaining the application's relevance and effectiveness in the marketplace.

References

[1] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*, vol. 5. New York, NY, USA: Springer Science & Business Media, 2012.